A Comparative Study of the Semi-Elastic and Fully-Elastic MapReduce Models

Xiaoyong Xu and Maolin Tang
School of Electrical Engineering and Computer Science
Queensland University of Technology
2 George Street, Brisbane, QLD 4001, Australia
seanalex86@gmail.com

***Abstract*—MapReduce, which was initially proposed to handle big data in a cluster of computers, is becoming a popular programming model for big data processing in cloud computing. When MapReduce is used in cloud computing where everything is a service and the quality of service is important, a new issue that must be addressed is how to ensure a MapReduce computation will finish before a deadline in a dynamically changing cloud computing environment while minimizing its computation cost. The original MapReduce model cannot address the issue as it is not elastic, that is, it does not support adding resources to a MapReduce computation duration the runtime. To overcome the drawback of the original MapReduce model, a fully-elastic MapReduce is proposed in this paper. In addition, in this paper we study the performance of the fully-elastic model by comparing it with an existing model, namely, semi-elastic model, by theoretic analysis and by numerical experiments.**

***Keywords*—MapReduce; Cloud Computing; big data; elastic models;**

I. Introduction

Many commercial and scientific applications, such as data mining, bioinformatics, machine learning and web indexing, involve in processing massive amounts of data. MapReduce is a highly-popular programming model for such big data processing. MapReduce has the capability of processing terabytes and petabytes of data in a single job through parallelizing the job on a large-scale cluster of computing nodes, including one master node and a set of slave nodes. The master node assigns the map and reduce tasks of a job to those slave nodes and monitors the job's progress. Every salve node may process several map and reduce tasks.

There are three activities in the MapReduce model: *mapping*, *shuffling* and *reducing* phases. During the map phase, the map slots process the map tasks and generate the output. The map output will be partitioned into several regions, according to the number of reduce tasks. Once a map task is finished, the i^{th} region of the map output will be transferred to the i^{th} reduce slots. The process of transferring these output is shuffling. Once the shuffle phase ends, the reduce phase starts, all map output is sorted and merged, and then processed by the reduce slots. Finally, the reduce output is produced and the job is finished.

Cloud computing is a promising computing paradigm where computing resources can be delivered to the end users via the Internet as a service. The end users can subscribe the service in a pay-as-you-go way, while the service provider must ensure the quality of service (i.e. deadline) is satisfied. When MapReduce is applied in cloud computing, it is a service, thus it must be finished before the deadline, so that the quality of service can be satisfied. Meanwhile, the amount of resources for executing MapReduce should be minimized for the sake of cost-saving. However, it is a challenging problem for the original MapReduce model [1] which does not support adding resources to accelerate the job's progress during the job run-time on demand, because in cloud computing, which is a volatile computing environment, any performance degradation on the computing nodes or network probably delays the job's progress and finally leads to the deadline violation.

To overcome the drawback in the original MapReduce model, elastic MapReduce models [2][3][4] have been developed. All of them are semi-elastic MapReduce models, ones supporting scaling up resources for the execution of the map tasks, not for the reduce tasks. However, the work on finding a fully-elastic model, which has the ability of adding resources both for the map and reduce tasks execution, has been rarely studied.

In this paper, we will firstly propose a new MapReduce model called fully-elastic MapReduce, and then theoretically analyze the minimal amount of resources needed to increase to make sure the job can be finished before the deadline under both the semi-elastic and fully-elastic MapReduce models. Then, we will design 50 cases in term of job type, intervention timing and deadline pressure, and compare the two catagories of models in these cases by looking at the minimal amount of additional resources and the number of cases where the deadline is violated.

The rest of the paper is organized as follows: Section II discusses the related work; Section III reviews a semi-elastic MapReduce model and presents our fully-elastic MapReduce model; Section IV theoretically analyze performance of the semi-elastic and fully-elastic MapReduce models; in Section V, we compare the semi-elastic and fully MapReduce models by empirical study; and finally Section VI concludes the study.

II. Related Work

The original MapReduce model [1] is not elastic during the job execution, the resources cannot be added to speed

up the progress even when the job is under the high risk of deadline violation. Some MapReduce models are implemented on top of clouds, such as Amazon Elastic MapReduce [5] and Azure MapReduce [6], allows the users to request new resources on demand between different jobs, but not during the job run-time. Furthermore, some works [7][8] bring the elasticity to the data flow of MapReduce, breaking the barriers between map and reduce stages and making it applicable in incremental computation and iterative execution. However, all these MapReduce models cannot support the addition of new resources as the job is underway. The static feature of these models probably leads to the deadline violation when the job is executed under a volatile environment like cloud computing.

In order to overcome the drawback in the original MapReduce, DEMLMA [2], a Hadoop-like MapReduce, was proposed, which allowed for node count to be increased, speeding up a job without terminating the currently running task. Unlike DEMLMA which was designed for the physical environment, Cardosa et al. [3] developed an elastic model called STEAMEngine which was implemented under the virtualized environment. In this model, the job was executed by the virtual machines and the number of these machines could increase if the users requested. AbdelBaky et al. [4] studied the MapReduce under the hybrid environment, and proposed MapReduce-CometCloud. In this model, users could request for more resources from the public clouds during the job run-time when the local resources could not meet the demand. Although all these models support adding resources during the job execution, only the map phase, rather than the reduce phase, is sped up due to the additional resources, thus they are categorized as semi-elastic MapReduce. The resource utilization in these models are low. When these models are used under the cloud computing where the resources are not free, lower resource utilization means more costly for running MapReduce.

However, few works study the fully-elastic elastic models which supports adding resources both for the execution of the map and reduce tasks. Furthermore, the comparison between the semi-elastic and fully-elastic MapReduce has not been studied.

III. Overview of the Semi-Elastic and Fully-Elastic MapReduce Models

A. The Semi-Elastic MapReduce

The semi-elastic MapReduce model supports adding computing nodes during the job execution to accelerate the job's progress when the job falls behind the schedule. However, it just reduces the duration of the map phase, rather than the reduce phase.

An example of the semi-elastic MapReduce model is presented in Fig. 1. In this cluster, there are two map slots and two reduce slots. The number of map tasks is 5 while the number of reduce tasks is 2. After two map tasks are finished, a node holding one map slot and one reduce slot is added, then the rest three map tasks will be completed by three map slots in one round. If the node is not added, the rest three map tasks will be executed by two map slots and finished in two rounds. Therefore, the duration of the map phase is reduced. The region number of the map output (or the number of reduce tasks) has not been changed, still amounting to 2, thus the third reduce slot on the new node is idle and has no task to process. Therefore, the reduce phase is not reduced.

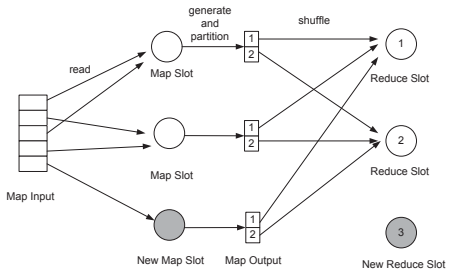

Fig. 1. The execution process of the semi-elastic MapReduce model

In the semi-elastic MapReduce model, the reduce phase starts intermediately when the shuffle phase is finished. New nodes are not allowed to add after this phase starts. When the reduce phase starts, all map tasks are finished. If new nodes are added at this time, they will have no tasks to process, so the duration of map phase cannot be decreased. Therefore, the situation that the nodes are added after the reduce phase starts is not discussed in this model.

B. The Fully-Elastic MapReduce

Here we give a new elastic MapReduce model, called fully-elastic MapReduce, which also supports adding computing nodes during the job execution to accelerate the job's progress when the job falls behind the schedule. Unlike the semi-elastic one, through adding new nodes, this model not just reduces the duration of the map phase, but also the duration of reduce phase.

An example of the fully-elastic MapReduce model is presented in Fig. 2. In this cluster, there are two map slots and two reduce slots. The number of map tasks is 5 while the number of reduce tasks is 2. At beginning, two map slots respectively one map task, the output is partitioned into 2 regions, amounting to the number of reduce slots, and then the map output will be transferred to the corresponding reduce slots. Then, one node holding one map slot and one reduce slot is added, and three map slots will process the rest three map tasks and finish them in one round. Thus, just like in the semi-elastic MapReduce model, the duration of the map phase is also reduced.

Unlike the semi-elastic one, the fully-elastic MapReduce model introduces a repartition algorithm based on Consistent Hashing (CH), to repartition the map output after the new nodes are added. A outstanding feature of CH is that it guarantees *Monotonicity*, which means the data only can be moved to a new partition from the original partitions, but not between original buckets. Furthermore, CH also supports the concept of virtual partitions, which are allocated to one actual partition just like in Dynamo. Through the application

of virtual partitions, the data can be redistributed under the new partitions in balance. Through adopting this repartition algorithm, the output of the rest three map tasks will be partitioned into 3 regions evenly, thus the number of reduce tasks is increased to 3. Then, the third region will be transferred to the new reduce slot. Furthermore, the output of the first two map tasks, having been transferred to the reduce slots, also need to be repartitioned by CH. After the repartition, the data belonging to the third region will be transferred to the third reduce slot. The process of repartitioning and transferring the map output under the original regions is called *Repartition* phase.

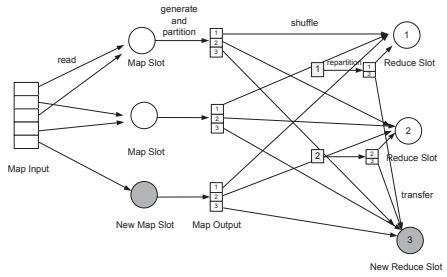

Fig. 2. The execution process of the fully-elastic MapReduce model

In the fully-elastic MapReduce model, the reduce phase only starts when both the shuffle phase and repartition phase are finished. Furthermore, like in the semi-elastic MapReduce model, new nodes are also not allowed to add once the reduce phase starts. After the repartition, the total size of output has not been changed while the number of reduce tasks is decreased, thus the input size of each reduce task is decreased. Since the reduce tasks are completed by the three reduce slots in one round, the duration of reduce phase is reduced.

IV. Theoretical Study

When MapReduce is used in cloud computing, a new issue is how to ensure a MapReduce job will finish before a deadline while using as less resources (computing nodes) as possible. Therefore, for the semi-elastic and fully-elastic MapReduce models, a resource provision problem must be addressed is how to ensure the deadline will not be violated by adding a minimal amount of computing nodes, which is formulated as:

$$min \qquad n^* - n \tag{1}$$

$$s.t: \qquad t + T^{rest} \leq t_d \tag{2}$$

$$n^* > n \tag{3}$$

where n^* is the variable needed to be minimized, representing the total number of nodes including the additional ones, n is a constant, denoting the number of the nodes (not including the master node) initially provided to run the job, thus $n^* - n$ denotes the number of the additional nodes. These nodes are homogeneous which means they have the same computing power. Moreover, the first constraint means the job must be completed before the deadline, where t is the instant of adding nodes, and T^{rest} is the duration of completing the rest part of the job, t_d is the deadline. The second constraint indicates that the number of additional nodes must be larger than 0.

Then, we will respectively calculate the minimal number of the nodes needed to add to ensure the job will be finished before the deadline under the semi-elastic and fully-elastic MapReduce models. But before this, we will respectively estimate the duration T^{rest} of the rest part of the job.

A. Time Estimation of the Rest Part of the Job

Let T^{rest1} be the completion time of the rest part of job under the semi-elastic MapReduce model. It consists of three parts:

$$T^{rest1} = T^{map} + T^{nol} + T^{red1} \tag{4}$$

where T^{map} is the duration of the rest map phase, T^{nol} is the duration of the non-overlapped part of the shuffle phase with the map phase and T^{red1} is the duration of the reduce phase. Since the situation that adding nodes after the reduce phase starts is not discussed in the semi-elastic MapReduce model, T^{rest1} contains the duration of the whole reduce phase.

By the instant t, m_1 map tasks have been completed while m_2 ones remain to be completed, the total number of map tasks is m and $m = m_1 + m_2$. For simplicity, we consider the default configuration in Hadoop: each node has the equal number (denoted as k) of map and reduce slots [9]. Thus the rest map tasks will be executed by kn^* map slots. They will be competed in $\lceil \frac{m_2}{kn^*} \rceil$ rounds. I and v^m are two constants, respectively denoting the input size of each map task and the data size each map slot can process per time unit, then the duration of the rest map phase is $T^{map} = \lceil \frac{m_2}{kn^*} \rceil \frac{I}{v^m}$, where $\frac{I}{v^m}$ means the duration of each round, and it equals to the duration of one map tasks as all map slots have the same computing power.

Once one round of map tasks are finished, their output will be transferred to the reduce slots. We assume that each map task produces the output with the same ratio (ρ) to its input, so the output size of each map task is ρI. Let c be the size of the data transferred per time unit. Transferring each round of the output data is overlapped with processing the next round of map tasks. Let T^o be the non-overlapped duration of each round of map tasks, and $T^o = \max\left\{\left(\frac{\rho I}{c} - \frac{I}{v^m}\right), 0\right\}$.

Specially, the last round of map tasks are not overlapped as all map tasks have been finished. Thus, the total duration of the non-overlapped shuffle phase is

$$T^{nol} = \left(\left\lceil \frac{m_1}{kn} \right\rceil + \left\lceil \frac{m_2}{kn^*} \right\rceil - 1\right) T^o + \frac{\rho I}{c} \tag{5}$$

Furthermore, we assume the computation complexity of reduce tasks is $O(x \cdot log(x))$ where x is the size of reduce input, because sorting algorithms are integrated in this phase [10]. Thus, the duration of the reduce phase is $T^{red1} = \frac{\rho I m}{knv^r} \log \frac{\rho I m}{kn}$, where v^r is a constant denoting the data size each reduce slot can process per time unit, kn is the number of reduce tasks, equaling to that of reduce slots.

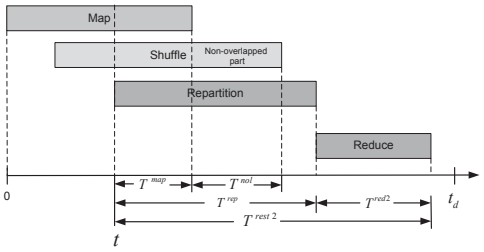

Fig. 3. The completion time of the rest part of job under the fully-elastic MapReduce model

Let T^{rest2} be the completion time of the rest part of job after adding new nodes with repartition. As Fig. 3 shows, the reduce phase starts when both the shuffle and repartition phases completes, so T^{rest2} is expressed as

$$T^{rest2} = \max\left\{T^{map} + T^{nol}, T^{rep}\right\} + T^{red2} \tag{6}$$

where T^{rep} is the duration of the repartition phase.

The duration of repartition phase, expressed by Eq. (7), consists of two parts: (1) the duration of repartitioning the output under the original partitions whose size is $\frac{\rho Im_1}{kn}$; (2) the duration of transferring the repartitoned data to the new slots, whose size is $\frac{\rho Im_1}{k}\left(\frac{1}{n} - \frac{1}{n^*}\right)$.

$$T^{rep} = \frac{\rho Im_1}{knv^p} + \frac{\rho Im_1}{kc}\left(\frac{1}{n} - \frac{1}{n^*}\right) \tag{7}$$

After the repartition, the input size of each reduce task is decreased to $\frac{\rho Im}{kn^*}$, thus the duration of reduce phase is $T^{red2} = \frac{\rho Im}{kn^*v^r}\log\frac{\rho Im}{kn^*}$.

B. The Minimal Number of the Nodes Needed to Add

Having presented the time estimation of the rest part of the job, we will respectively investigate the optimal solution (denoted as n_1) to the resource provision problem under the semi-elastic MapReduce model and the optimal solution (denoted as n_2) under the fully-elastic MapReduce model.

The detailed process of calculating n_1 is presented as follow: Firstly, Let T^{rest1} replace T^{rest} in the first constraint, then

$$\left\lceil\frac{m_2}{kn^*}\right\rceil\frac{I}{v^m} + \left(\left\lceil\frac{m_1}{kn}\right\rceil + \left\lceil\frac{m_2}{kn^*}\right\rceil - 1\right)T^o + \frac{\rho I}{c} + T^{red1} \le t_d - t$$

For any positive number x, $x \le \lceil x\rceil < x+1$. To make it more convenient to manipulate the inequalities, we discuss them in the best scenario where $\lceil x\rceil = x$. Then, the lower interger bound of n^* can be found, expressed as follow:

$$n_1 = \left\lceil\frac{\frac{m_2 I}{kv^m} + \frac{m_2 T^o}{k}}{t_d - t - (\frac{m_1}{kn} - 1)T^o - \frac{\rho I}{c} - T^{red1}}\right\rceil \tag{8}$$

Specially, if $n_1 \le n$, there is no solution to the resource provision problem, in other words, the semi-elastic model cannot ensure the job will meet the deadline under this situation. Consequently, the minimal number of the nodes needed to add to ensure the job will be finished before the deadline is $n_1 - n$.

On the other hand, the detailed process of calculating n_2 is presented as follow: Firstly, let T^{rest2} replace T^{rest} in the first constraint, then

$$\Leftrightarrow \quad t + T^{map} + T^{nol} + T^{red2} - t_d \le 0 \tag{9}$$

$$\text{AND } t + T^{rep} + T^{red2} - t_d \le 0 \tag{10}$$

Let n_{21} and n_{22} be the minimal values of n^* in Ineq. (9) and (10), making these inequalities true. Then,

$$n_2 = \max\{n_{21}, n_{22}\} \tag{11}$$

For the sake of fairness, we discuss the best scenario where $\lceil x\rceil = x$ for a positive x. Then, replace n^* by x, and the left part of Ineq. (9) can be rewritten as

$$f(x) = A + \frac{B}{x} + \frac{\rho Im}{v^r kx}\log\frac{\rho Im}{kx} \tag{12}$$

where $x \ge n$, $A = (\frac{m_1}{kn} + 1)T^o + \frac{I}{v^m} + \frac{\rho I}{c} + t - t_d$ and $B = \frac{m_2 I}{v^m} + m_2 T^o$. $f(x)$ is a continuous function. Then, Ineq. (9) can be rewritten as $f(n^*) \le 0$

Especially, $f(n) > 0$, since the job cannot be finished before the deadline if no new nodes are added. Then,

$$f^{'}(x) = -\frac{1}{k^2x^2}\left(B + \frac{\rho Im}{v^r}\log\frac{\rho Im}{kx} + \frac{\rho Im}{v^r\ln a}\right)$$

Obviously, $\forall x \ge n$, $f^{'}(x) < 0$. Thus, there must exist a value x_o ($x_o > n$), making $f(x_o) = 0$, and when $x \ge x_o$, $f(x) \le 0$. x_o can be calculated easily by *Bisection* method. Since n_{21} is an integer, $n_{21} = \lceil x_o\rceil$.

Similarly, replace n^* by x, and then the left part of the Ineq. (10) can be rewritten as

$$g(x) = C - \frac{\rho Im_1}{ckx} + \frac{\rho Im}{v^r kx}\log\frac{\rho Im}{kx} \tag{13}$$

where $C = \frac{\rho Im_1}{knv^p} + \frac{\rho Im_1}{knc} + t - t_d$. $g(x)$ is a continuous function. Then, Ineq. (10) can be rewritten as $g(n^*) \le 0$ Then,

$$g^{'}(x) = -\frac{1}{k^2x^2}\left(\frac{\rho Im}{v^r}\log\frac{\rho Im}{kx} + \frac{\rho Im}{v^r\ln a} - \frac{\rho Im_1}{c}\right)$$

If $\frac{m}{v^r\ln a} \ge \frac{m_1}{c}$, then $\forall x \ge n$, $g^{'}(x) < 0$. In addition, if $g(n) \le 0$, $\forall x \ge n$, $g(x) \le 0$, thus, $n_{22} = n + 1$; otherwise, there must be a value x_1 ($x_1 > n$) making $g(x_1) = 0$, and when $x \ge x_1$, $g(x) \le 0$. x_1 can be calculated easily by the bisection method. Since the minimal value n_{22} making $g(x) \le 0$ is an integer, $n_{22} = \lceil x_1\rceil$.

If $\frac{m}{v^r\ln a} < \frac{m_1}{c}$, then there must be a value x_2 ($x_2 \ge n$), making $g^{'}(x) = 0$. x_2 can be calculated easily by the bisection method. When $n \le x < x_2$, $g^{'}(x) < 0$, while when $n \le x > x_2$, $g^{'}(x) > 0$. Therefore, the minimal value of $g(x)$ is $g(x_2)$. If $g(x_2) > 0$, there is no value making $g(x) \le 0$; in other words, n_{22} does not exist, it means the fully-elastic model cannot ensure avoid the violation of the deadline under this situation.

If $g(x_2) \le 0$, and $g(n) \le 0$, $\forall n \le x \le x_2$, $g(x) \le 0$, thus, $n_{22} = n + 1$; otherwise, if $g(x_2) \le 0$, and $g(n) > 0$,

there must be a value x_3 ($n < x_3 \leq x_2$) making $g(x_3) = 0$, and when $x \geq x_3$, $g(x) \leq 0$. x_3 can be calculated easily by the bisection method. Since the minimal value n_{22} making $g(x) \leq 0$ is an integer, $n_{22} = \lceil x_3 \rceil$.

Consequently, having calculated the optimal solution n_2 to the resource provision problem under the fully-elastic MapReduce model, we get the minimal number of the nodes needed to add to ensure the job will be finished before the deadline, which is $n_2 - n$.

V. Empirical Study

This section will study the semi-elastic MapReduce model and the fully-elastic MapReduce model by case study. In the case study, we will design 50 presentive cases and compare the performance of the two models in the 50 cases.

A. Cases

The semi-elastic and fully-elastic MapReduce models will be compared under 50 cases. These cases are equally divided into two groups: *map-heavy* cases and *reduce-heavy* cases. In the map-heavy cases, the jobs are the map-heavy ones, which need more computation power for the map activity than for the reduce activity. Grep is an example of such map-heavy MapReduce jobs. On the other hand, in the reduce-heavy cases, the jobs are the reduce-heavy ones, which need more computation time for the reduce activity than for the map activity. One typical example of reduce-heavy MapReduce jobs is SequenceCount.

In each group, the cases are characterized by their *intervention timing* and *deadline pressure* (5 × 5). The intervention timing (denoted as $I.T.$) is the percentage of the completed map tasks in the total map tasks, which is given by

$$I.T. = \frac{m_1}{m} \times 100\% \tag{14}$$

The cases are divided into five levels of $I.T.$: 20%, 40%, 60%, 80% and 100%, respectively. The higher level of $I.T.$, the more map tasks have been completed.

On the other hand, the deadline pressure (denoted as $D.P.$) indicates the percentage of the minimal time needed to fill to ensure the deadline will not be violated in the time (denoted as T) of completing the job without additional nodes, which is expressed as

$$D.P. = \frac{T - t_d}{T} \times 100\% \tag{15}$$

In this equation, T is a constant and it can be easily calculated according to Eq. (4) where $m_1 = m$, $m_2 = 0$ and $n^* = n$. Similarly, the cases are divided into five levels of $D.P.$: 10%, 20%, 30%, 40% and 50%, respectively. The higher level of $D.P.$ means the shorter period from the time of adding new nodes to the deadline.

B. Parameters Setting

We will set the same parameters for the semi-elastic and fully-elastic MapReduce models in each case. Part of the parameters are given in Table I.

TABLE I
The parameters in the models

Parameters	Map-heavy cases	Reduce-heavy cases
v_m	2	20
v_r	20	10
r	0.1	1.5
m	600	600
n	10	10
k	2	2
c	5	5
v_p	10	10

With regard to the rest parameters, in each case, according to Eq. (14), $m_1 = m \times I.T.$, and $m_2 = m - m_1$; according to Eq. (15), $t_d = (1 - D.P.) \times T$. Furthermore, for simplicity, the instant of adding nodes, t is configured as the completion time of m_1 map tasks, thus $t = \lceil \frac{m_1}{nk} \rceil \frac{I}{v^m}$.

C. Numerical Results and Analysis

In each case, given the parameters, the minimal numbers of the nodes to add to ensure the job will meet the deadline under the semi-elastic and fully-elastic MapReduce models are respectively calculated by the methods presented in the previous section. The numerical results under the map-heavy cases are given in Table II, while the numerical results under the reduce-heavy cases in Table III. In the tables, the first and second number of the bivector respectively denote the minimal amount of nodes needed to add under the semi-elastic and fully-elastic MapReduce model. Specially, N means the deadline cannot be met under an elastic model.

TABLE II
The numerical results in the map-heavy cases

Deadline Pressure ($D.P.$)	Intervention Timing ($I.T.$)				
	20%	40%	60%	80%	100%
10%	(2,2)	(3,3)	(4,4)	(11,8)	(N,N)
20%	(4,4)	(6,5)	(11,9)	(N,N)	(N,N)
30%	(7,6)	(11,10)	(37,24)	(N,N)	(N,N)
40%	(11,10)	(24,19)	(N,N)	(N,N)	(N,N)
50%	(19,16)	(69,42)	(N,N)	(N,N)	(N,N)

As seen in Table II, for the map-heavy jobs, in the cases with the early intervention or low deadline pressure, the minimal amount of nodes needed to add under the semi-elastic MapReduce model is slightly larger than that under the fully-elastic MapReduce model. Especially in the cases where $D.P. = 10\%$ and $I.T. = 20\%, 40\%, 60\%$ or $D.P. = 20\%$ and $I.T. = 20\%$, the minimal numbers under the two models are the same. As the level of $I.T$ or $D.P$ increases, the gap in the minimal amount of nodes needed to add between the models gradually widen. For example, in the cases where $I.T. = 40\%$, when $D.P. = 10\%$, the two models need to add the same amount of nodes, but when $D.P. = 50\%$, the semi-elastic model has to add 27 more nodes than the fully-elastic one. However, when $I.T.$ or $D.P.$ increases to a relative high level, both the models cannot work on ensuring the job can be finished before the deadline by adding new nodes. In detail, in the cases where $I.T. = 60\%$ and $D.P. \geq 40\%$, $I.T. = 80\%$

and $D.P. \geq 40\%$ or $I.T. = 100\%$ and $D.P. \geq 10\%$, they cannot work anymore. With regard to the total number of these cases where deadline cannot be met, the two models have the same figures, both amounting to 11.

TABLE III
THE NUMERICAL RESULTS IN THE REDUCE-HEAVY CASES

Deadline Pressure ($D.P$)	Intervention Timing ($I.T$)				
	20%	40%	60%	80%	100%
10%	(6,2)	(9,2)	(22,2)	(N,2)	(N,2)
20%	(22,3)	(104,3)	(N,4)	(N,4)	(N,5)
30%	(N,5)	(N,5)	(N,6)	(N,7)	(N,13)
40%	(N,7)	(N,8)	(N,10)	(N,15)	(N,N)
50%	(N,11)	(N,13)	(N,16)	(N,106)	(N,N)

On the other hand, as Table III shows, for the reduce-heavy jobs, the minimal amount of nodes needed to add under the fully-elastic MapReduce model is much less than that under the semi-elastic MapReduce model. For instance, for the map-heavy jobs, the semi-elastic model needs to add 27 nodes more than the fully-elastic one at most, but for the reduce-heavy jobs, this figure increases to 101. On the other hand, under the same deadline pressure, the semi-elastic MapReduce model needs earlier intervention than the fully-elastic one. For example, in the cases where $D.P. = 20\%$, under the semi-elastic model, the nodes must be added before $I.T = 60\%$, while under the fully-elastic model, even $I.T = 100\%$, which means all map tasks have been completed, the deadline still can be met once 5 nodes are added. Meanwhile, when the intervention is conducted at the same time, the semi-elastic MapReduce model cannot ensure the deadline will be met under a higher deadline pressure than the fully-elastic MapReduce model. For instance, in the cases where $I.T. = 20\%$, if $D.P. \geq 30\%$, the semi-elastic MapReduce model cannot ensure the deadline will be caught, but even if $D.P. = 50\%$, the fully-elastic MapReduce model still can ensure that. However, if the intervention is much late as well as the deadline pressure is greatly high ($I.T. = 100\%$ and $D.P. \geq 40\%$), both the two models cannot ensure the deadline will not be violated.

VI. CONCLUSION AND FUTURE WORK

In this paper, we have proposed a fully-elastic MapReduce model and analyzed the performance of the fully-elastic MapReduce models, by comparing it with an existing semi-elastic MapReduce model. Moreover, We have deducted the minimal number of nodes needed to add in order to meet the pre-defined deadline for both the semi-elastic model and the fully-elastic model.

In addition, we have conducted an empirical study of the two MapReduce models by case study. The case study results have shown that the fully-elastic MapReduce model outperformed the semi-elastic MapReduce model for all the studied cases. In particular, for the map-heavy jobs, the fully-elastic MapReduce model outperformed the semi-elastic MapReduce model slightly. If the intervention was early or the deadline pressure was low, compared with the semi-elastic MapReduce model, the fully-elastic MapReduce model just added slightly less or even same nodes to avoid missing the deadline. On the contrary, if the intervention was too late or the deadline pressure was too high, the deadline would not be caught even by adding new nodes. Under this situation, the total numbers of the cases where the deadline was not met under the two models were the same. On the other hand, for the reduce-heavy jobs, the fully-elastic MapReduce model outperformed the semi-elastic MapReduce model greatly. On one hand, the fully-elastic MapReduce model added much less nodes than the semi-elastic MapReduce model to ensure the deadline was caught. On the other hand, compared with the semi-elastic MapReduce model, the fully-elastic MapReduce model ensured the deadline was met by adding nodes in the cases with the later intervention timing or the higher level of deadline pressure, reflecting its more flexibility on intervention timing and stronger endurance on deadline pressure. Besides these, we respectively found the cases where neither of the semi-elastic and fully-elastic MapReduce models ensured the deadline was met for map-heavy and reduce-heavy jobs.

Due to the time limitation, we only considered job type, invention timing and deadline pressure in our case study. In the future, we will do a more comprehensive case study considering more metrics, such as communication rate and repartition rate.

ACKNOWLEDGMENT

This research was funded by the State Scholarship Fund of the China Scholarships Council (CSC) and the CSC Top-Up Scholarship of Queensland University of Technology, Australia.

REFERENCES

[1] J. Dean and S. Ghemawat, "MapReduce: simplified data processing on large clusters," *Commun. ACM*, vol. 51, no. 1, pp. 107–113, Jan. 2008.

[2] Z. Fadika and M. Govindaraju, "Delma: Dynamically elastic MapReduce framework for cpu-intensive applications," in *Cluster, Cloud and Grid Computing (CCGrid), 2011 11th IEEE/ACM International Symposium on*, 2011, pp. 454–463.

[3] M. Cardosa, P. Narang, A. Chandra, H. Pucha, and A. Singh, "Steamengine: Driving MapReduce provisioning in the cloud," in *High Performance Computing (HiPC), 2011 18th International Conference on*, 2011, pp. 1–10.

[4] M. AbdelBaky, H. Kim, I. Rodero, and M. Parashar, "Accelerating MapReduce analytics using cometcloud," in *Cloud Computing (CLOUD), 2012 IEEE 5th International Conference on*, 2012, pp. 447–454.

[5] "Amazon elastic MapReduce." [Online]. Available: http://aws.amazon.com/elasticMapReduce/

[6] T. Gunarathne, T.-L. Wu, J. Qiu, and G. Fox, "MapReduce in the clouds for science," in *Cloud Computing Technology and Science (CloudCom), 2010 IEEE Second International Conference on*, 2010, pp. 565–572.

[7] P. Bhatotia, A. Wieder, R. Rodrigues, U. A. Acar, and R. Pasquin, "Incoop: MapReduce for incremental computations," in *Proceedings of the 2nd ACM Symposium on Cloud Computing*, ser. SOCC '11. New York, NY, USA: ACM, 2011, pp. 7:1–7:14.

[8] Y. Zhang, Q. Gao, L. Gao, and C. Wang, "iMapReduce: A distributed computing framework for iterative computation," *Journal of Grid Computing*, vol. 10, no. 1, pp. 47–68, 2012.

[9] T. White, *Hadoop: The Definitive Guide*. O'Reilly Media, 2009.

[10] J.Berliska and M. Drozdowski, "Scheduling divisible MapReduce computations," *Journal of Parallel and Distributed Computing*, vol. 71, no. 3, pp. 450 – 459, 2011.